Rank-profile revealing Gaussian elimination and the CUP matrix decomposition

Claude-Pierre Jeannerod[a], Clément Pernet[b], Arne Storjohann[c]

[a] *INRIA, Laboratoire LIP (CNRS, ENS de Lyon, INRIA, UCBL), Université de Lyon, 46, allée d'Italie, F-69364 Lyon Cedex 07, France*
[b] *INRIA, Laboratoire LIG (CNRS, Grenoble INP, INRIA, UJF, UPMF) ENSIMAG Antenne de Montbonnot, 51, ave Jean Kuntzmann, F-38330 Montbonnot Saint-Martin, France*
[c] *David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada, N2L 3G1*

Abstract

Transforming a matrix over a field to echelon form, or decomposing the matrix as a product of structured matrices that reveal the rank profile, is a fundamental building block of computational exact linear algebra. This paper surveys the well known variations of such decompositions and transformations that have been proposed in the literature. We present an algorithm to compute the `CUP` decomposition of a matrix, adapted from the `LSP` algorithm of Ibarra, Moran and Hui (1982), and show reductions from the other most common Gaussian elimination based matrix transformations and decompositions to the `CUP` decomposition. We discuss the advantages of the `CUP` algorithm over other existing algorithms by studying time and space complexities: the asymptotic time complexity is rank sensitive, and comparing the constants of the leading terms, the algorithms for computing matrix invariants based on the `CUP` decomposition are always at least as good except in one case. We also show that the `CUP` algorithm, as well as the computation of other invariants such as transformation to reduced column echelon form using the `CUP` algorithm, all work in place, allowing for example to compute the inverse of a matrix on the same storage as the input matrix.

Email addresses: `claude-pierre.jeannerod@ens-lyon.fr` (Claude-Pierre Jeannerod), `clement.pernet@imag.fr` (Clément Pernet), `astorjoh@uwaterloo.ca` (Arne Storjohann)

Keywords: Gaussian elimination, LU matrix decomposition, echelon form, reduced echelon form, rank, rank profile, fast linear algebra, in place computations

1. Introduction

Gaussian elimination and the corresponding matrix decompositions, such as the `LU` decomposition $A = LU$ as the product of lower triangular L and upper triangular U, are fundamental building blocks in computational linear algebra that are used to solve problems such as linear systems, computing the rank, the determinant and a basis of the nullspace of a matrix. The `LU` decomposition, which is defined for matrices whose leading principal minors are all nonsingular, can be generalized to all nonsingular matrices by introducing pivoting on one side (rows or columns), leading to the `LUP` or `PLU` decomposition, P a permutation matrix. Matrices with arbitrary dimensions and rank can be handled by introducing pivoting on both sides, leading to the `LQUP` decomposition of Ibarra et al. (1982) or the `PLUQ` decomposition (Golub and Van Loan, 1996; Jeffrey, 2010), Q a second permutation matrix. We recall the precise definitions of these decompositions in Section 2. For now, we note that the decompositions are not unique. In particular, they can differ depending on the pivoting strategy used to form the permutations P and Q. Whereas in numerical linear algebra (Golub and Van Loan, 1996) pivoting is used to ensure a good numerical stability, good data locality, and reduce the fill-in, the role of pivoting differs in the context of exact linear algebra. Indeed, only certain pivoting strategies for these decompositions will reveal the rank profile of the matrix (see Section 2), which is crucial information in many applications using exact Gaussian elimination, such as Gröbner basis computations (Faugère, 1999) and computational number theory (Stein, 2007).

In this article we consider matrices over an arbitrary field and analyze algorithms by counting the required number of arithmetic operations from the field. Many computations over $\mathbb{Z}$ or $\mathbb{Q}$ (including polynomials with coefficients from these rings) reduce to linear algebra over finite fields using techniques such as linearization or homomorphic imaging (e.g., reduction modulo a single prime, multi-modular reduction, p-adic lifting). Applications such as cryptanalysis make intensive use of linear algebra over finite fields directly. Unlike arithmetic operations involving integers, the cost of an

operation in a finite field does not depend on the value of its operands, and counting the number of field operations is indicative of the actual running time.

The asymptotic time complexity of linear algebra of matrices over a field has been well studied. While Gaussian elimination can be performed on an $n \times n$ matrix using $O(n^3)$ arithmetic operations, a subcubic complexity of $O(n^\omega)$ with $\omega < 2.38$ the exponent of matrix multiplication (see von zur Gathen and Gerhard, 2003, §12.1) is obtained using Bunch and Hopcroft's (1974) block recursive algorithm for the `LUP` decomposition. Algorithms for computing the `LSP` and `LQUP` decompositions of rank deficient matrices are given by Ibarra et al. (1982). As an alternative to decomposition, Keller-Gehrig (1985) adapts some earlier work by Schönhage (1973) to get an algorithm that computes a nonsingular transformation matrix X such that AX is in column echelon form. For an $m \times n$ matrix such that $m \leq n$, all of the Gaussian elimination based algorithms just mentioned have an $O(nm^{\omega-1})$ time complexity.

Only recently has the complexity of computing a rank profile revealing decomposition or a transformation to echelon form been studied in more details: algorithms with a rank-sensitive time complexity of $O(mnr^{\omega-2})$ for computing a transformation to echelon and reduced echelon forms are given by Storjohann and Mulders (1998) and Storjohann (2000); similar rank-sensitive time complexities for the `LSP` and `LQUP` decompositions of Ibarra et al. (1982) have been obtained by Jeannerod (2006). On the one hand, while offering a rank-sensitive time complexity, the algorithms in (Storjohann and Mulders, 1998; Storjohann, 2000; Jeannerod, 2006) are not necessarily in-place. On the other hand, Dumas et al. (2008) describe an in-place variant of the `LQUP` decomposition but, while analyzing the constant in the leading term, does not achieve a rank-sensitive complexity.

The aim of this article is to gather, generalize, and extend these more refined analyses to the most common Gaussian elimination based matrix decompositions. We propose an algorithm computing a new matrix decomposition, the `CUP` decomposition, and show reductions to it for all common matrix decompositions. We assess that, among all other matrix decompositions, the algorithm for `CUP` is the preferred Gaussian elimination algorithm to be used as a building block routine for the following reasons.

1. We show how all other decompositions can be recovered from the `CUP` decomposition. Furthermore, the time complexities for all algorithms

computing the alternative decompositions and transformations (considering the constant in the leading term) are never better than the proposed reduction to `CUP`, except by a slight amount in one case.
2. The complexity of the algorithm for `CUP` is rank sensitive (whereas no such result could be produced with some of the other algorithms).
3. The reduction to `CUP` allows us to perform these computations in-place, that is, with essentially no more memory than what the matrix products involved already use.
4. The `CUP` decomposition offers the best modularity: combined with other classic routines like `TRSM`, `TRTRI`, `TRULM` and `TRLUM` (which shall be recalled in the course of the article), it allows to compute solutions to other linear algebra problems such as rank, rank profile, nullspace, determinant and inverse with the best time complexities.

1.1. Notation and definitions

Matrix entries are accessed using zero-based indexing: for example, $a_{0,1}$ denotes the entry lying on the first row and second column of the matrix $A = [a_{i,j}]$. In order to make the description of our algorithms simpler, we adopt the following convention (Pilgrim, 2004): intervals of indices include the lower bound but exclude the upper bound, that is,

$$a..b = \{a, a+1, \ldots, b-1\}.$$

Our algorithms make heavy use of conformal block decompositions of matrices. In the literature, new variable names are typically used for each block. Instead, we refer to blocks using intervals of indices in order to emphasize the fact that no memory is being allocated and that our algorithms actually work in-place. The notation $A_{a..b}^{c..d}$ represents the submatrix of A formed by the intersection of rows of index from a to $b-1$, and the columns of index from c to $d-1$. For example, a 2×2 block decomposition of an $m \times n$ matrix A shall be referred to as

$$A = \left[\begin{array}{c|c} A_{0..k}^{0..\ell} & A_{0..k}^{\ell..n} \\ \hline A_{k..m}^{0..\ell} & A_{k..m}^{\ell..n} \end{array}\right]$$

for suitable integers k and ℓ. Similarly, $A_a^{c..d}$ denotes the subrow of row a of A whose column indices range from c to $d-1$.

Following Ibarra et al. (1982), we say a rectangular matrix $A = [a_{i,j}]$ is *upper triangular* if $a_{i,j} = 0$ for $i > j$, and that it is *lower triangular* if its

transpose A^T is upper triangular. A matrix that is either upper or lower triangular is simply called *triangular*, and if in addition $a_{i,i} = 1$ for all i then it is called *unit triangular*. For two $m \times n$ matrices L and U such that $L = [\ell_{i,j}]$ is unit lower triangular and $U = [u_{i,j}]$ is upper triangular, we shall use the notation $[L\backslash U]$ to express the fact that L and U are stored one next to the other within the same $m \times n$ matrix. Thus, $A = [L\backslash U]$ is the $m \times n$ matrix $A = [a_{i,j}]$ such that $a_{i,j} = \ell_{i,j}$ if $i > j$, and $a_{i,j} = u_{i,j}$ otherwise.

For a permutation $\sigma : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$, the associated permutation matrix $P = [P_{i,j}]_{0 \le i,j < n}$ is defined by $P_{i,\sigma(i)} = 1$ and $P_{i,j} = 0$ for $j \neq \sigma(i)$. Multiplying a matrix on the left by P applies the permutation σ to the rows of that matrix, while multiplying on the right by the transpose $P^T = P^{-1}$ applies σ to its columns. We denote by $T_{i,j}$ the permutation matrix that swaps indices i and j and leaves the other elements unchanged.

1.2. Organization of the article

Section 2 reviews and exposes the links between the most common matrix decompositions originating from Gaussian elimination: `LU`, `LUP`, `Turing-Decomposition`, `LSP`, `LQUP`, and `QLUP`, together with our variant of the `LSP` decomposition, the `CUP` decomposition. Section 3 gives algorithms to compute all these matrix decompositions with no extra memory allocation. More precisely, we give an algorithm for computing a `CUP` decomposition and show how all of the other decompositions can be derived from it. The advantages of the `CUP` algorithm compared to other existing algorithms for Gaussian elimination are discussed in Section 4, based on an analysis of time and space complexities. In Section 5 we comment on the `PLE` decomposition, which is the row-counterpart of the `CUP` decomposition, and we conclude in Section 6.

2. Review of Gaussian elimination based matrix decompositions

Throughout this section, let A be an $m \times n$ matrix with rank r. The *row rank profile* of A is the lexicographically smallest sequence of r row indices $i_0 < i_1 < \cdots < i_{r-1}$ such that the corresponding rows of A are linearly independent. The matrix A is said to have *generic rank profile* if its first r leading principal minors are nonzero.

2.1. `LU` based decompositions

If A has generic rank profile it has a unique `LU` decomposition: $A = LU$ for L an $m \times m$ unit lower triangular matrix with last $m - r$ columns those

of the identity, and U an $m \times n$ upper triangular matrix with last $m-r$ rows equal to zero. In our examples, zero entries of a matrix are simply left blank, possibly nonzero entries are indicated with $*$, and necessarily nonzero entries with $\overline{*}$.

Example 1. The LU decomposition of a 6×4 matrix A with rank 3.

$$\overset{A}{\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}} = \overset{L}{\begin{bmatrix} 1 & & & \\ * & 1 & & \\ * & * & 1 & \\ * & * & * & 1 \\ * & * & * & \\ * & * & * & \end{bmatrix}} \overset{U}{\begin{bmatrix} \overline{*} & * & * & * \\ & \overline{*} & * & * \\ & & \overline{*} & * \\ & & & \\ & & & \end{bmatrix}}$$

Uniqueness of U and the first r rows of L follows from the generic rank profile condition on A, while uniqueness of the last $m-r$ rows of L follows from the condition that L has last $m-r$ columns those of the identity matrix. If A has row rank profile $[0, 1, \ldots, r-1]$, then it only takes a column permutation to achieve generic rank profile; for such input matrices, Aho et al. (1974, §6.4) and Bunch and Hopcroft (1974) give a reduction to matrix multiplication for computing an LUP decomposition: $A = LUP$, with P a permutation matrix and $AP^T = LU$ the LU decomposition of AP^T. Allowing row and column permutations extends the LU decomposition to any matrix, without restriction on the rank profile. In the context of numerical computation, the row and column permutations leading to the best numerical stability are chosen (see for example Golub and Van Loan, 1996, §3.4), and are referred to as partial pivoting (e.g., column permutations only) and complete pivoting (column and row permutations). In the context of symbolic computation, a key invariant of the input matrix that should be revealed is the rank profile. In the next subsection we recall the most common matrix decompositions based on the column echelon form of A, thus all revealing the row rank profile of A.

2.2. Rank profile revealing decompositions

Let A be an $m \times n$ input matrix with row rank profile $[i_0, \ldots, i_{r-1}]$. Recall how the iterative version of the Gaussian elimination algorithm transforms A to column echelon form. The algorithm detects the row rank profile of A during the elimination. For $i = i_0, \ldots, i_{r-1}$, a pivoting step (interchange of two columns) is performed, if required, to ensure the pivot entry in row i_j and column j is nonzero, and then entries to the right of the pivot are eliminated.

By recording the column swaps separately in a permutation matrix P, and the eliminations in a unit upper triangular matrix U, we arrive at a structured transformation of the input matrix to column echelon form AP^TU.

Example 2. The following shows the structured transformation of a 7×5 matrix A with row rank profile $[0, 3, 4]$ to column echelon form C.

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} \overset{U}{\begin{bmatrix} 1 & * & * & * & * \\ & 1 & * & * & * \\ & & 1 & * & * \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} = \overset{C}{\begin{bmatrix} \overline{*}_1 & & & & \\ * & & & & \\ * & & & & \\ * & \overline{*}_2 & & & \\ * & * & \overline{*}_3 & & \\ * & * & * & & \\ * & * & * & & \end{bmatrix}}$$

Once an echelon form C has been obtained, post-multiplication by a diagonal matrix D can be used to make the pivot elements in the echelon form equal to 1, and a further post-multiplication by a unit lower triangular L can be used to eliminate entries before each pivot, giving the canonical reduced column echelon form R of A.

Example 3. The following shows the structured transformation of the matrix from Example 2 to reduced column echelon form.

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} \overset{U}{\begin{bmatrix} 1 & * & * & * & * \\ & 1 & * & * & * \\ & & 1 & * & * \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} \overset{D}{\begin{bmatrix} \overline{*}_1^{-1} & & & & \\ & \overline{*}_2^{-1} & & & \\ & & \overline{*}_3^{-1} & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} \overset{L}{\begin{bmatrix} 1 & & & & \\ * & 1 & & & \\ * & * & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} = \overset{R}{\begin{bmatrix} 1 & & & & \\ * & & & & \\ * & & & & \\ & 1 & & & \\ & & 1 & & \\ * & * & * & & \\ * & * & * & & \end{bmatrix}}$$

Proposition 1. Let A be an $m \times n$ matrix with row rank profile $[i_0, \ldots, i_{r-1}]$. Corresponding to any $n \times n$ permutation matrix P such that the submatrix of AP^T comprised of rows $i_0 \ldots, i_{r-1}$ has generic rank profile, there exists

- a unique $n \times n$ unit upper triangular matrix

$$U = \begin{bmatrix} U_1 & U_2 \\ & I_{n-r} \end{bmatrix}$$

such that $C = AP^TU$ is a column echelon form of A, and

- a unique diagonal matrix $D = \text{Diag}(D_1, I_{n-r})$ and $n \times n$ unit lower triangular
$$L = \begin{bmatrix} L_1 & \\ & I_{n-r} \end{bmatrix}$$

such that $R = AP^T UDL$ is the reduced column echelon form of A.

The literature contains a number of well known matrix decompositions that reveal the row rank profile: the common ingredient is a permutation matrix P that satisfies the requirements of Proposition 1. The Turing decomposition of the transpose of A is as shown in Example 3 except with the matrices U, D, L, and P inverted and appearing on the right-hand side of the equation, and was introduced by Corless and Jeffrey (1997) as a generalization to the rectangular case of the square decomposition $A = LDU$ given by Turing in his seminal 1948 paper. The `LSP` and `LQUP` decompositions are due to Ibarra et al. (1982). A more compact variant of `LQUP` is `QLUP`, and the `CUP` decomposition is another variation of the `LSP` decomposition that we introduce here. The `LQUP` and `QLUP` decompositions also involve a permutation matrix Q such that the first r rows of QA are equal to rows $i_0, \ldots, i_{r-1}$ of A. Once the permutations P and Q satisfying these requirements are fixed, these five decompositions are uniquely defined and in a one-to-one correspondence. The following proposition links these decompositions by defining each of them in terms of the matrices U, C, D, L, and R of Proposition 1. For completeness, the proposition begins by recalling the definitions of the classic transformations of A to column echelon form and to reduced column echelon form.

Proposition 2. Corresponding to an $n \times n$ permutation matrix P such that rows $i_0, \ldots, i_{r-1}$ of AP^T have generic rank profile, let U, C, D, L and R be the matrices defined in Proposition 1. The following transformations exist and are uniquely defined based on the choice of P.

- `ColEchTrans`: (P, U, C) such that $AP^T U = C$.
  Example:
$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} \overset{U}{\begin{bmatrix} 1 & * & * & * & * \\ & 1 & * & * & * \\ & & 1 & * & * \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} = \overset{C}{\begin{bmatrix} \overline{*}_1 & & \\ * & & \\ * & & \\ * & \overline{*}_2 & \\ * & * & \overline{*}_3 \\ * & * & * \\ * & * & * \end{bmatrix}}$$

- `RedColEchTrans`: (P, X, R) such that $AP^T X = R$ with $X = UDL$.
  Example:

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} \overset{X}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} = \overset{R}{\begin{bmatrix} 1 & & & & \\ * & & & & \\ * & & & & \\ & 1 & & & \\ & & & 1 & \\ * & * & * & & \\ * & * & * & & \end{bmatrix}}$$

Now let $\overline{L} = L^{-1}, \overline{D} = D^{-1}$ and $\overline{U} = U^{-1}$. The following decompositions exist and are uniquely defined based on the choice of P.

- `TuringDecomposition`: $(R, \overline{L}, \overline{D}, \overline{U}, P)$ such that $AP^T = R\overline{LDU}$.
  Example:

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} = \overset{R}{\begin{bmatrix} 1 & & & & \\ * & & & & \\ * & & & & \\ & 1 & & & \\ & & & 1 & \\ * & * & * & & \\ * & * & * & & \end{bmatrix}} \overset{\overline{L}}{\begin{bmatrix} 1 & & & & \\ * & 1 & & & \\ * & * & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} \overset{\overline{D}}{\begin{bmatrix} \overline{*}_1 & & & & \\ & \overline{*}_2 & & & \\ & & \overline{*}_3 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}} \overset{\overline{U}}{\begin{bmatrix} 1 & * & * & * & * \\ & 1 & * & * & * \\ & & 1 & * & * \\ & & & 1 & \\ & & & & 1 \end{bmatrix}}$$

- `CUP`: $(C, \overline{U}, P)$ such that $AP^T = C\overline{U}$.
  Example:

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} = \overset{C}{\begin{bmatrix} \overline{*}_1 & & & \\ * & & & \\ * & & & \\ * & \overline{*}_2 & & \\ * & * & \overline{*}_3 & \\ * & * & * & \\ * & * & * & \end{bmatrix}} \overset{\overline{U}}{\begin{bmatrix} 1 & * & * & * & * \\ & 1 & * & * & * \\ & & 1 & * & * \\ & & & 1 & \\ & & & & 1 \end{bmatrix}}$$

- `LSP`: (L', S, P) such that $A = L'SP$ with
  - L' an $m \times m$ unit lower triangular with columns $i_0, \ldots, i_{r-1}$ equal to columns $0, \ldots, r-1$ of CD, and other columns those of I_m.
  - S an $m \times n$ semi upper triangular matrix with rows $i_0, \ldots, i_{r-1}$ equal to rows $0, \ldots, r-1$ of $\overline{DU}$, and other rows zero.

Example:

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} = \overset{L'}{\begin{bmatrix} 1 & & & & & & \\ * & 1 & & & & & \\ * & & 1 & & & & \\ * & & & 1 & & & \\ * & & & * & 1 & & \\ * & & & * & * & 1 & \\ * & & & * & * & & 1 \end{bmatrix}} \overset{S}{\begin{bmatrix} \overline{*}_1 & * & * & * & * \\ & & & & \\ & \overline{*}_2 & * & * & * \\ & & \overline{*}_3 & * & * \\ & & & & \\ & & & & \end{bmatrix}}$$

In addition to P, let Q be an $m \times m$ permutation matrix such that rows $0, \ldots, r-1$ of QA are rows $i_0, \ldots, i_{r-1}$ of A. Then the following decompositions exist and are uniquely defined based on the choice of P and Q.

- `LQUP`: (L', Q, U', P) such that $A = L'QU'P$, with L' the same matrix as in the `LSP` decomposition, and U' an $m \times n$ matrix with first r rows equal to the first r rows of $\overline{DU}$, and last $n - r$ rows zero.
  Example:

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} = \overset{L'}{\begin{bmatrix} 1 & & & & & & \\ * & 1 & & & & & \\ * & & 1 & & & & \\ * & & & 1 & & & \\ * & & & * & 1 & & \\ * & & & * & * & 1 & \\ * & & & * & * & & 1 \end{bmatrix}} Q \overset{U'}{\begin{bmatrix} \overline{*}_1 & * & * & * & * \\ & \overline{*}_2 & * & * & * \\ & & \overline{*}_3 & * & * \\ & & & & \\ & & & & \end{bmatrix}}$$

- `QLUP`: (Q, L'', U'', P) with L'' an $m \times r$ unit lower triangular matrix equal to the first r columns of $Q^T CD$, and U'' an $r \times n$ upper triangular matrix equal to the first r rows of $\overline{DU}$.
  Example:

$$\overset{AP^T}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}} = Q \overset{\overline{L}}{\begin{bmatrix} 1 & & & \\ * & 1 & & \\ * & * & 1 & \\ * & & & \\ * & & & \\ * & * & * & \\ * & * & * & \end{bmatrix}} \overset{\overline{U}}{\begin{bmatrix} \overline{*}_1 & * & * & * & * \\ & \overline{*}_2 & * & * & * \\ & & \overline{*}_3 & * & * \end{bmatrix}}$$

Note that the `LSP`, `LQUP`, and `QLUP` decompositions can be transformed from one to the other without any field operations.

3. The CUP matrix decomposition: algorithm and reductions

In this section we propose an algorithm for the CUP decomposition, and show how each of the other four decompositions and two transformations of Proposition 2 can be recovered via the CUP decomposition. For clarity we postpone to Section 4 the discussion of the advantages, in terms of time and space complexities, of the CUP decomposition algorithm over other Gaussian elimination based agorithms in the literature.

All algorithms presented in this section are recursive and operate on blocks; this groups arithmetic operations into large matrix multiplication updates of the form $C = \alpha AB + \beta C$. Reducing dense linear algebra to matrix multiplication not only leads to a reduced asymptotic time complexity because of the subcubic exponent of matrix multiplication, but also ensures good efficiency in practice thanks to the availability of highly optimized subroutines (see for example Dumas et al., 2002).

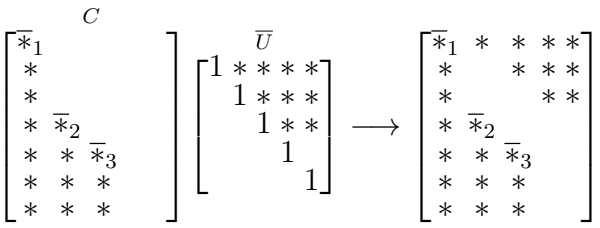
3.1. Space-sharing storage and in-place computations

Dealing with space complexity, we make the assumption that a field element as well as indices use an atomic memory space. We distinguish two ways to reduce the memory consumption of our algorithms:

Space-sharing storage. All the algorithms described here, except for matrix multiplication, store their matrix outputs (excluding permutations) in the space allocated to their inputs. For example, when solving the triangular system with matrix right-hand side $UX = B$, the input matrix B will be overwritten by the output matrix $X = U^{-1}B$. In the case of matrix decompositions, the output consists of one or two permutations that can be stored using an amount of space linear in the sum of the matrix dimensions, and two structured matrices that can be stored together within the space of the input matrix. We call this a space-sharing storage.

The space-sharing storage for the CUP and ColEchTrans of a matrix of rank r stores the first r columns of the lower triangular factor below the first r rows of the upper triangular factor in the same matrix. Overlap is avoided by storing only the nontrivial diagonal entries. Here is an example for a CUP

decomposition.



The `RedColEchelonTrans` can be stored in a space-sharing manner up to some permutation. Indeed, any transformation to reduced column echelon form of rank r can be written as

$$AP^T \begin{bmatrix} X_1 & X_2 \\ & I_{n-r} \end{bmatrix} = R = Q^T \begin{bmatrix} I_r \\ \underline{R} \; 0 \end{bmatrix}$$

for some permutation matrix Q. Up to this permutation of rows, this allows the space-sharing storage

$$\begin{bmatrix} X_1 \; X_2 \\ \underline{R} \end{bmatrix}.$$

Similar space-sharing storage formats are possible for the `QLUP`, `LSP`, `LQUP` and `TuringDecomposition`.

In-place computation. More importantly, we will also focus on the intermediate memory allocations of the algorithms presented. As the algorithms rely heavily on matrix multiplication, one needs to take into account the memory allocations in the matrix multiplication algorithm. Using the classical $O(n^3)$ algorithm, one can perform the operation $C \leftarrow \alpha AB + \beta C$ with no additional memory space beyond that required to store the input and output, but this is no longer the case with Strassen and Winograd's $O(n^{2.81})$ algorithms (for a detailed analysis see for example Huss-Lederman et al., 1996; Boyer et al., 2009). Consequently, we will consider matrix multiplication as a black-box operation, and analyze the memory complexity of our algorithms independently of it. This leads us to introduce the following definition.

Definition 1. A linear algebra algorithm is called *in-place* if it does not involve more than $O(1)$ extra memory allocations for field elements for any of its operations except possibly in the course of matrix multiplications.

In particular, when *classical* matrix multiplication is used, in-place linear algebra algorithms only require a constant amount of extra memory allocation.

3.2. Basic building blocks: `MM`, `TRSM`, `TRMM`, `TRTRI`, `TRULM`, `TRLUM`

We assume that Algorithm 1 for matrix multiplication is available. We

Algorithm 1: `MM`(A, B, C, α, β)

Data: A an $m \times \ell$ matrix.
Data: B an $\ell \times n$ matrix.
Data: C an $m \times n$ matrix that does not overlap with either A or B.
Data: α a scalar.
Data: β a scalar.
Ensures: $C \leftarrow \alpha AB + \beta C$.

also use the well-known routines `TRSM` and `TRTRI`, defined in the level 3 BLAS legacy (Dongarra et al., 1990) and the LAPACK library (Anderson et al., 1999). The `TRSM` routine simultaneously computes several linear system solutions X from an invertible triangular matrix A and a matrix right-hand side B. The matrix A can be lower or upper triangular, unit or non-unit triangular, left-looking ($AX = B$) or right-looking ($XA = B$). Algorithm 2 below illustrates the case "right-looking, upper, non-unit," and shows how to incorporate matrix multiplication; the algorithm is clearly in-place (Dumas et al., 2008, §4.1) and has running time $O(\max\{m, n\}\, n^{\omega-1})$.

Algorithm 2: `TRSM`(Right, Up, NonUnit, U,B)

Data: U an $n \times n$ invertible upper triangular matrix.
Data: B an $m \times n$ matrix.
Ensures: $B \leftarrow BU^{-1}$.

```
1 begin
2     if n = 1 then
3         for i ← 0, ..., m − 1 do b_{i,0} ← b_{i,0}/u_{0,0}
4     else
5         k ← ⌊n/2⌋
          /* B = [B_1 B_2] with B_1 m × k and U = [U_1 V; 0 U_2] with U_1 k × k */
6         TRSM(Right, Up, NonUnit, U_{0..k}^{0..k}, B_{0..m}^{0..k})   /* B_1 ← B_1 U_1^{-1} */
7         MM(B_{0..m}^{0..k}, U_{0..k}^{k..n}, B_{0..m}^{k..n}, −1, 1)   /* B_2 ← B_2 − B_1 V */
8         TRSM(Right, Up, NonUnit, U_{k..n}^{k..n}, B_{0..m}^{k..n})   /* B_2 ← B_2 U_2^{-1} */
```

We remark that an algorithm similar to Algorithm 2 can be written for the so-called `TRMM` routine, which computes the product of a triangular matrix by an arbitrary matrix, in the same eight variants (for the square case see for example Dumas et al., 2008, §6.2.1). Clearly, `TRMM` has the same in-place and complexity features as `TRSM`.

The `TRTRI` routine inverts a triangular matrix that can be either upper or lower triangular, unit or non-unit triangular. Algorithm 3 illustrates the case "upper, non-unit" and shares the following features with the three other variants: it reduces to matrix multiplication via two calls to `TRSM` in half the dimension, has a cost in $O(n^\omega)$, and is in-place.

Algorithm 3: `TRTRI`(Up, NonUnit, U)

Data: U an $n \times n$ invertible upper triangular matrix.
Ensures: $U \leftarrow U^{-1}$.

begin
  if $n = 1$ **then**
    $u_{0,0} \leftarrow 1/u_{0,0}$
  else
    $k \leftarrow \lfloor \frac{n}{2} \rfloor$
    /* $U = \begin{bmatrix} U_1 & V \\ & U_2 \end{bmatrix}$ with U_1 $k \times k$, s.t. $U^{-1} = \begin{bmatrix} U_1^{-1} & -U_1^{-1}VU_2^{-1} \\ & U_2^{-1} \end{bmatrix}$ */
    `TRSM`(Right, Up, NonUnit, $U_{k..n}^{k..n}, U_{0..k}^{k..n}$) /* $V \leftarrow VU_2^{-1}$ */
    `TRSM`(Left, Up, NonUnit, $U_{0..k}^{0..k}, U_{0..k}^{k..n}$) /* $V \leftarrow U_1^{-1}V$ */
    $U_{0..k}^{k..n} \leftarrow -U_{0..k}^{k..n}$ /* $V \leftarrow -V$ */
    `TRTRI`($U_{0..k}^{0..k}$) /* $U_1 \leftarrow U_1^{-1}$ */
    `TRTRI`($U_{k..n}^{k..n}$) /* $U_2 \leftarrow U_2^{-1}$ */

We conclude this section by introducing the `TRULM` and `TRLUM` routines: given a unit lower triangular matrix L and an upper triangular matrix U stored one next to the other within the same square matrix, they return the products UL and LU, respectively. Unlike `TRSM` and `TRTRI`, these routines are neither BLAS nor LAPACK routines, and to our knowledge have not yet been described elsewhere. Algorithms 4 and 5 show how they can be implemented in-place and at cost $O(n^\omega)$. The routine `TRULM` is the key step that enables us to compute the reduced echelon form in-place and in subcubic time (see Algorithm 8), while `TRLUM` will be used to derive a fast and in-place method for matrix products of the form $B \leftarrow A \times B$ (see Algorithm 9).

Algorithm 4: `TRULM` (A)

Data: $A = [L\backslash U]$ an $n \times n$ matrix.
Ensures: $A \leftarrow UL$.

```
1 begin
2     if n > 1 then
3         k ← ⌊n/2⌋
          /* A = [L1\U1  U2; L2  L3\U3] and LU = [X1 X2; X3 X4] with L1,U1,X1 k×k */
4         TRULM(A_{0..k}^{0..k})                                  /* X1 ← U1 L1 */
5         MM(A_{0..k}^{k..n}, A_{k..n}^{0..k}, A_{0..k}^{0..k}, 1, 1)   /* X1 ← X1 + U2 L2 */
6         TRMM(Right, Low, Unit, A_{k..n}^{k..n}, A_{0..k}^{k..n})      /* X2 ← U2 L3 */
7         TRMM(Left, Up, NonUnit, A_{k..n}^{k..n}, A_{k..n}^{0..k})     /* X3 ← U3 L2 */
8         TRULM(A_{k..n}^{k..n})                                  /* X4 ← U3 L3 */
```

Algorithm 5: `TRLUM` (A)

Data: $A = [L\backslash U]$ an $n \times n$ matrix.
Ensures: $A \leftarrow LU$.

```
1 begin
2     if n > 1 then
3         k ← ⌊n/2⌋
          /* A = [L1\U1  U2; L2  L3\U3] and UL = [X1 X2; X3 X4] with L1,U1,X1 k×k */
4         TRLUM(A_{k..n}^{k..n})                                  /* X4 ← L3 U3 */
5         MM(A_{k..n}^{0..k}, A_{0..k}^{k..n}, A_{k..n}^{k..n}, 1, 1)   /* X4 ← X4 + L2 U2 */
6         TRMM(Left, Low, Unit, A_{0..k}^{0..k}, A_{0..k}^{k..n})       /* X2 ← L1 U2 */
7         TRMM(Right, Up, NonUnit, A_{0..k}^{0..k}, A_{k..n}^{0..k})    /* X3 ← L2 U1 */
8         TRLUM(A_{0..k}^{0..k})                                  /* X1 ← L1 U1 */
```

3.3. The `CUP` matrix decomposition algorithm

We now present Algorithm 6, a block recursive algorithm for factoring any $m \times n$ matrix A as $A = CUP$ with C a column echelon form revealing the row rank profile of A, U a unit upper triangular matrix, and P a permutation matrix. It is a variation on the `LSP` and `LQUP` decomposition algorithms of Ibarra et al. (1982), similar to the ones presented by Jeannerod (2006) and Dumas et al. (2008). Similar to the basic building blocks of Section 3.2, the description using submatrix indices shows that the entire algorithm can be performed in-place: each recursive call transforms a block into a block of the form $[C\backslash U]$, and the remaining operations are a `TRSM`, a matrix multiplication, and applying row and column permutations. The `CUP` algorithm also handles the case where $m > n$. The ability to work in-place and handle matrices of arbitrary shape are the main advantages of the algorithm for `CUP` decomposition over the `LSP` or `LQUP` of Ibarra et al. (1982), as will be discussed in Section 4.

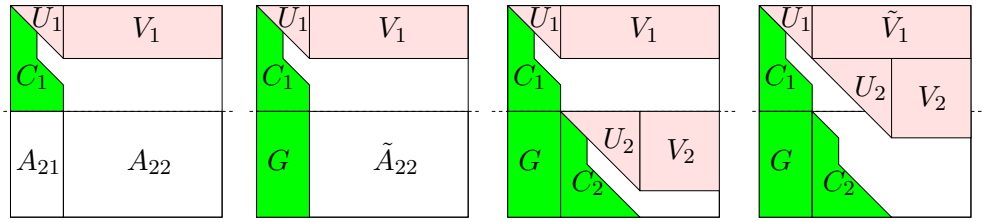


Figure 1: Illustration of Algorithm 6: steps 12, 18, 24, and 27.

Figure 1 shows the state of the input matrix after the four main steps (steps 12, 18, 24, and 27). At step 27, the zero rows between blocks V_1 and $[U_2\, V_2]$ are shifted down.

Proposition 3. Algorithm 6 computes a `CUP` decomposition of the input matrix A together with its rank r and row rank profile $[i_0, \ldots, i_{r-1}]$.

Proof. It is clear from the description of the algorithm that P is a permutation matrix, U is unit upper triangular, and C is in column echelon form. To check that $A = CUP$, one can proceed exactly as Jeannerod (2006, §3.1). □

Algorithm 6: `CUP`(A)

Data: A an $m \times n$ matrix.
Result: $(r, [i_0, \ldots, i_{r-1}], P)$.
Ensures: $A \leftarrow [C \backslash U]$ such that, embedding U in an $n \times n$ unit upper triangular matrix makes $A = CUP$ a `CUP` decomposition of A, and r and $[i_0, \ldots, i_{r-1}]$ are the rank and the row rank profile of A.

begin
  if $m = 1$ **then**
    if $A \neq 0$ **then**
      $i \leftarrow$ column index of the first nonzero entry of A
      $P \leftarrow T_{0,i}$, the transposition of indices 0 and i
      $A \leftarrow AP$
      for $i \leftarrow 2 \ldots n$ **do** $A_{0,i} \leftarrow A_{0,i} A_{0,0}^{-1}$
      return $(1, [0], P)$
    return $(0, [], I_n)$
  $k \leftarrow \lfloor \frac{m}{2} \rfloor$ `/*` $A = \left[\frac{A_1}{A_2}\right]$ `with` A_1 $k \times n$ `*/`
  $(r_1, [i_0, \ldots, i_{r_1-1}], P_1) \leftarrow \texttt{CUP}(A_{0..k}^{0..n})$ `/*` $A_1 \leftarrow C_1 [U_1\ V_1] P_1$ `*/`
  if $r_1 = 0$ **then**
    $(r_2, [i_0, \ldots, i_{r_2-1}], P_2) \leftarrow \texttt{CUP}(A_{k..m}^{0..n})$
    return $(r_2, [i_0 + k, \ldots, i_{r_2-1} + k], P_2)$
  $A_{k..m}^{0..n} \leftarrow A_{k..m}^{0..n} P_1^T$ `/*` $\left[\tilde{A}_{21}\ \tilde{A}_{22}\right] \leftarrow [A_{21}\ A_{22}]\, P_1^T$ `*/`
  `/*` $A = \begin{bmatrix} C_1 \backslash U_1 & V_1 \\ 0 & 0 \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix}$ `with` $C_1 \backslash U_1$ $k \times r_1$ `and` V_1 $r_1 \times (n - r_1)$ `*/`
  `TRSM`$(\text{Right}, \text{Upper}, \text{Unit}, A_{0..r_1}^{0..r_1}, A_{k..m}^{0..r_1})$ `/*` $G \leftarrow A_{21} U_1^{-1}$ `*/`
  if $r_1 = n$ **then**
    return $(r_1, [i_0, \ldots, i_{r_1-1}], P_1)$
  `MM`$(A_{k..m}^{0..r_1}, A_{r_1..n}^{0..r_1}, A_{k..m}^{r_1..n}, -1, 1)$ `/*` $H \leftarrow A_{22} - G V_1$ `*/`
  $(r_2, [j_0, \ldots, j_{r_2-1}], P_2) \leftarrow \texttt{CUP}(A_{k..m}^{r_1..n})$ `/*` $H \leftarrow C_2 U_2 P_2$ `*/`
  $A_{0..r_1}^{r_1..n} \leftarrow A_{0..r_1}^{r_1..n} P_2^T$ `/*` $\tilde{V}_1 \leftarrow V_1 P_2^T$ `*/`
  for $j \leftarrow r_1 \ldots r_1 + r_2$ **do**
    $A_j^{n..j} \leftarrow A_j^{n..j+k-r_1}$ `/* Moving` U_2, V_2 `up next to` V_1 `*/`
    $A_{j..m}^{j+k-r_1} \leftarrow [0]$
  $P \leftarrow \text{Diag}(I_{r_1}, P_2) P_1$
  return $(r_1 + r_2, [i_0, \ldots, i_{r_1-1}, j_0 + k, \ldots, j_{r_2-1} + k], P)$

3.4. `ColEchTrans` transormation via a `CUP` decomposition

Proposition 4 states how to compute the required transformation matrix X such that $AX = C$ is in echelon form.

Proposition 4. Let $A = CUP$ be the `CUP` decomposition of an $m \times n$ matrix A of rank r computed with Algorithm 6. Let X be the inverse of U embedded in an $n \times n$ unit upper triangular matrix. Then $AP^TX = E$ is a transformation of A to column echelon form.

PROOF. One need only verify that $CX^{-1} = CUP = A$. □

Algorithm 7: `ColEchTrans`(A)

Data: A an $m \times n$ matrix over a field.
Result: $(r, [i_0, \ldots, i_{r-1}], P)$ where P is a permutation matrix.
Ensures: $A \leftarrow [C \backslash X]$ such that, embedding X in an $n \times n$ unit upper triangular matrix makes $AP^TX = C$ a transformation of A to column echelon form, $r = \text{rank}(A)$ and $[i_0, \ldots, i_{r-1}]$ is the row rank profile of A.

```
1 begin
2     (r, [i_0, ..., i_{r-1}], P) ← CUP(A)
      /* A = [C\U_1 U_2; 0] where U_1 is r × r upper triangular   */
3     TRSM(Left, Upper, Unit, A[0..r, 0..r], A[0..r, r..n])      /* M ← U_1^{-1} U_2 */
4     A[0..r, r..n] ← −A[0..r, r..n]                             /* N ← −M */
5     TRTRI(Upper, A[0..r, 0..r])                                /* U ← U^{-1} */
6     return (r, [i_0, ..., i_{r-1}], P)
```

Algorithm 7 shows how the computation of the column echelon form can be done in-place and reduces to that of `CUP`, `TRSM` and `TRTRI` and therefore has a time complexity of $O(mnr^{\omega-2})$.

3.5. `RedColEchTrans` transformation via `CUP` decomposition

Proposition 5 states how the reduced column echelon form can be computed from the `CUP` decomposition. Recall that $T_{i,j}$ denotes the permutation matrix that swaps indices i and j and leaves the other elements unchanged.

Proposition 5. Let $(r, [i_0, \ldots, i_{r-1}], P)$ and $[C\backslash Y]$ be the output of Algorithm 7 called on an $m \times n$ matrix A. As previously, assume that the matrix Y is embedded in an $n \times n$ unit upper triangular matrix. Let $Q = T_{0,i_0} T_{1,i_1} \ldots T_{r-1,i_{r-1}}$ so that $QC = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}$, where L_1 is an $r \times r$ non-singular lower triangular matrix. Let

$$R = C \begin{bmatrix} L_1^{-1} & \\ & I_{m-r} \end{bmatrix} \text{ and } X = Y \begin{bmatrix} L_1^{-1} & \\ & I_{m-r} \end{bmatrix}.$$

Then $AP^T X = R$ is a transformation of A to reduced column echelon form.

Proof. We start by showing that R is in reduced column echelon form. Recall that the permutation matrix Q is such that row k of a matrix A is row i_k of $Q^T A$. Hence for $j \in \{0 \ldots r-1\}$, row i_j of R is row j of the identity matrix I_m.

Consider column j of R. From the above remark, it has a 1 in row i_j. We now show that any coefficient above it is zero. Let $i < i_j$, then $R_{i,j} = \sum_{k=0}^{m} C_{i,k}[L^{-1}]_{k,j}$. As C is in column echelon form and $i < i_j$, $C_{k,i} = 0$ for any $k \geq j$. Since L^{-1} is lower triangular, $L^{-1}_{k,j} = 0$ for any $k < j$, hence $R_{i,j} = 0$.

It remains to verify

$$AX = CUPP^T Y \begin{bmatrix} L_1^{-1} & \\ & I_{m-r} \end{bmatrix} = R.$$

□

Algorithm 8 shows how the computation of the reduced column echelon form transformation can be done in-place and reduces to that of `CUP`, `TRSM`, `TRTRI`, and `TRULM` and therefore has a time complexity of $O(mnr^{\omega-2})$.

4. Discussion

Because it is the building block of dense linear algebra algorithms, Gaussian elimination appears in many different forms in the literature and in software implementations. Softwares are mostly based on either a transformation to echelon form, or one of the decompositions of Proposition 2, from which the usual computations such as linear system solving, determinant, rank, rank profile, nullspace basis, etc. can be derived. We discuss here the

Algorithm 8: `RedColEchTrans`(A)

Data: A an $m \times n$ matrix over a field.
Result: $(r, [i_0, \ldots, i_{r-1}], P)$ where P is a permutation matrix.
Ensures: $A \leftarrow \begin{bmatrix} X_1 & X_2 \\ R_2 & 0 \end{bmatrix}$ such that if $X = \begin{bmatrix} X_1 & X_2 \\ & I_{n-r} \end{bmatrix}$ and $Q = T_{0,i_0} T_{1,i_1} \ldots T_{r-1,i_{r-1}}$ and $R = Q^T \begin{bmatrix} I_r \\ R_2 \; 0 \end{bmatrix}$ then $AP^TX = R$ is a transformation of A to reduced column echelon form, $r = \text{rank}(A)$, and $[i_0, \ldots, i_{r-1}]$ is the column rank profile of A.

1 **begin**
2 $(r, [i_0, \ldots, i_{r-1}], P) \leftarrow$ `ColEchTrans`(A)
`/* Notations:` $A = [C \backslash T_1 \; T_2]$ `and` $X_2 \leftarrow T_2$. `*/`
3 **for** $j \leftarrow 0 \ldots r-1$ **do**
4 Swap$(A_j^{0..j}, A_{i_j}^{0..j})$ `/*` $\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \leftarrow QC$ `*/`
5 `TRSM`(Right, Lower, Unit, $A_{0..r}^{0..r}$, $A_{r..m}^{0..r}$) `/*` $R_2 \leftarrow L_2 L_1^{-1}$ `*/`
6 `TRTRI`(Lower, $A_{0..r}^{0..r}$) `/*` $N \leftarrow L_1^{-1}$ `*/`
7 `TRULM`$(A_{0..r}^{0..r})$ `/*` $X_1 \leftarrow T_1 N$ `*/`
8 **return** $(r, [i_0, \ldots, i_{r-1}], P)$

advantages of the `CUP` decomposition algorithm presented in the previous section compared to some other algorithms in the literature.

In our comparison we only consider algorithms that reduce to matrix multiplication. Indeed, reduction to matrix multiplication is the only way to benefit from both the best theoretical complexity and the practical efficiency of highly optimized implementations of the level 3 BLAS routines (Dumas et al., 2008).

To our knowledge, the algorithms achieving subcubic time complexity for elimination of rank deficient matrices are

- the `LSP` and `LQUP` algorithms of Ibarra et al. (1982) for computing an `LSP` and `LQUP` decomposition,
- the `Gauss` and `GaussJordan` algorithms (Storjohann, 2000, Algorithms 2.8 and 2.9) for computing an echelon and reduced echelon form transforms, and

- the `StepForm` algorithm (Keller-Gehrig, 1985; Bürgisser et al., 1997, §16.5) for computing an echelon form.

For the sake of completeness, we recall the `GaussJordan` and `StepForm` algorithms in Appendix A and Appendix B, together with an analysis of their complexity. Note that already in his seminal paper, Strassen (1969) proposed an $O(n^\omega)$ algorithm to compute the inverse of a matrix under strong genericity assumptions on the matrix. The `GaussJordan` algorithm can be viewed as a generalization: it inverts any nonsingular matrix without assumption, and computes the reduced echelon form and a transformation matrix in the case of singular matrices.

Note that all references to `QLUP` decomposition that we know of (namely (Golub and Van Loan, 1996, §3.4.9) and (Jeffrey, 2010)) do not mention any subcubic time algorithm, but as it can be derived as a slight modification of Algorithm 6, we will also take this variant into account for our comparison.

Our comparison of the various algorithms will be based on both space complexity (checking whether the algorithms are in-place) and a finer analysis of the time complexity which considers the constant of the leading term in the asymptotic estimate.

4.1. Comparison of the `CUP` decomposition with `LSP`, `LQUP`, `QLUP`

`LSP`. As discussed in Section 3.1, the `LSP` decomposition can not enjoy a space-sharing storage, unless its L matrix is compressed by a column permutatin Q. Now when designing a block recursive algorithm similar to Algorithm 6, one needs to compress the S matrix returned by the first recursive call, in order to use a `TRSM` update similar to operation 18. This requires to either allocate an $r \times r$ temporary matrix, or to do the compression by swapping rows of S back and forth. Instead, if the upper triangular matrix is kept compressed during the whole algorithm, this becomes an `LQUP` decomposition algorithm.

`LQUP`. Although `LQUP` decomposition can use a space sharing storage, the intermediate steps of a block recursive algorithm derived from Algorithm 6 would require additional column permutations on the L matrix to give it the uncompressed shape. Instead, if one chooses to compute the `LQUP` decomposition with a compressed L matrix, this really corresponds to the `CUP` decomposition, up to row and column scaling by the pivots.

QLUP. The QLUP decomposition can also be computed by an adaptation of Algorithm 6 where rows of the lower triangular matrix have to be permuted. Such an algorithm for the QLUP decomposition would then share the same advantages as the CUP decomposition algorithm but the following three reasons make the CUP decomposition preferable.

- The overhead of row permutations on the lower part of the matrix might become costly especially with sparse matrices.
- Part of the structure of the matrix C is lost when considering L, Q instead: in C, any coefficient above a pivot, in a non-pivot row is known to be zero by the echelon structure, whereas this same coefficient in L has to be treated as any other coefficient, and be assigned the zero value.
- It seems difficult to implement an efficient storage for the permutation Q (as can be done for P, using LAPACK storage of permutations). One could think of setting $Q = T_{0,i_0} T_{1,i_1} \dots T_{r-1,i_{r-1}}$ after the algorithm has completed, as it is done for Algorithm 8. However this permutation does not correspond to the permutation that was applied to the non-pivot rows of L during the process of the algorithm (call it $\tilde{Q}$). We could not find any subquadratic time algorithm to generate this permutation $\tilde{Q}$ from the two permutations $\tilde{Q}_1$ and $\tilde{Q}_2$ returned by the recursive calls.

The four matrix decompositions LSP, LQUP, PLUQ, and CUP are mathematically equivalent: they can all be computed by a dedicated algorithm with the same amount of arithemtic operations, and any conversion from one to another only involves permutations and pivot scaling. Among them, the CUP decomposition stands out as the most natural and appropriate one from the computational point of view: it can use a space-sharing storage and be computed in place and with the least amount of permutations.

4.2. A rank sensitive time complexity

As one may expect, the rank of the input matrix affects the time complexity of the CUP algorithm. For example, using a naive cubic-time algorithm for matrix multiplication the CUP decomposition requires $2mnr - (n+m)r^2 + \frac{2}{3}r^3$ field operations for an $m \times n$ input matrix of rank r.

Assuming a subcubic algorithm for matrix multiplication, the analysis in the literature for most Gaussian elimination algorithms is not rank sensitive.

For example, the running time of the `LSP` and `LQUP` algorithms (Ibarra et al., 1982) is only shown to be $O(m^{\omega-1}n)$, assuming $m \leq n$. Following the analysis of the `GaussJordan` algorithm (Storjohann, 2000, Algorithm 2.8), we give in Proposition 6 a rank sensitive complexity for Algorithm 6 computing the `CUP` decomposition of an input matrix of arbitrary shape. According to the reductions of Section 3, the rank sensitive complexity bound of Proposition 6 also holds for the computation of all other decompositions and transformations of Proposition 2.

Proposition 6. Algorithm 6 computes a `CUP` decomposition of an $m \times n$ matrix of rank r using $O(mnr^{\omega-2})$ field operations.

PROOF. Denote by $T_{\texttt{CUP}}(m, n, r)$ be the number of field operations required by Algorithm 6 for an $m \times n$ matrix A of rank r.

In the following, we assume without loss of generality that m is a power of 2. Following Storjohann (2000) we count a comparison with zero as a field operation. Then, when $r = 0$ (that is, A is the zero matrix), we have $T(m, n, r) = O(mn)$. As in the algorithm, let r_1 be the rank of A_1 and let r_2 be the rank of H. Then

$$T_{\texttt{CUP}}(m,n,r) = \begin{cases} O(n) & \text{if } m = 1, \\ T_{\texttt{CUP}}(\frac{m}{2}, n, 0) + T_{\texttt{CUP}}(\frac{m}{2}, n, r_2) + O(mn) & \text{if } r_1 = 0, \\ T_{\texttt{CUP}}(\frac{m}{2}, n, n) + O(\frac{m}{2}n^{\omega-1}) & \text{if } r_1 = n, \\ T_{\texttt{CUP}}(\frac{m}{2}, n, r_1) + T_{\texttt{CUP}}(\frac{m}{2}, n - r_1, r_2) + & \\ O(\frac{m}{2}r_1^{\omega-1}) + O(\frac{m}{2}(n - r_1)r_1^{\omega-2}) & \text{if } 0 < r_1 < n. \end{cases}$$

Consider the ith recursive level: the matrix is split row-wise into 2^i slices of row dimension $m/2^i$. We denote by $r_j^{(i)}$ the rank of each of these slices, indexed by $j = 0 \ldots 2^i - 1$. For example $r_1^{(i)} = \text{rank}(A_{0..\frac{m}{2^i}}^{0..n})$.

At the ith recursive level the total number of field operations is in

$$O\left(\frac{m}{2^i}\sum_{j=0}^{2^{i-1}}\left(r_{2j+1}^{(i)}\right)^{\omega-1} + \frac{m}{2^i}n\sum_{j=0}^{2^{i-1}}\left(r_{2j+1}^{(i)}\right)^{\omega-2}\right).$$

Since $\omega \geq 2$ we have

$$\sum_{j=0}^{2^{i-1}}\left(r_{2j+1}^{(i)}\right)^{\omega-1} \leq r^{\omega-1}.$$

Now, using the fact that $a^{\omega-2} + b^{\omega-2} \leq 2^{3-\omega}(a+b)^{\omega-2}$ for $2 < \omega \leq 3$, we also have

$$\sum_{j=0}^{2^{i-1}} \left(r_{2j+1}^{(i)}\right)^{\omega-2} \leq \left(2^{i-1}\right)^{3-\omega} r^{\omega-2}.$$

Therefore, we obtain

$$T_{\texttt{CUP}}(m,n,r) = \left(\sum_{i=1}^{\log_2 m} \frac{m}{2^i} r^{\omega-1} + \sum_{i=1}^{\log_2 m} nm \left(\frac{r}{2^i}\right)^{\omega-2} \right),$$

which for $\omega > 2$ is in $O(mnr^{\omega-2})$. □

We refer to Appendix B for a discussion on why the `StepForm` algorithm does not have a rank sensitive time complexity.

4.3. Space complexity

In the presentations of Algorithms 2—6 we exhibited the fact that no temporary storage was used. Consequently all of these algorithms, as well as `RedEchelon` (Algorithm 8), work in-place as per Definition 1. For a square and nonsingular input matrix, `RedEchelon` thus gives an in-place algorithm to compute the inverse.

For comparison, the `GaussJordan` algorithm involves products of the type $C \leftarrow A \times C$ (Algorithm 11, lines 17 and 23), which requires a copy of the input matrix C into a temporary storage in order to use a usual matrix multiplication algorithm. These matrix multiplication in lines 17 and 23 could be done in-place using Algorithm 9, but the leading constant in the time complexity of this version of `GaussJordan` increases to $3 + 1/3$ from 2 for $\omega = 3$.

We refer to Appendix B for a discussion on why the `StepForm` algorithm is not in-place.

4.4. Leading constants

For a finer comparison we compute the constant of the leading term in the complexities of all algorithms presented previously. For simplicity we assume that $m = n = r$ (i.e., the input matrix is square and nonsingular). The complexity of each algorithm is then of the form $K_\omega n^\omega + o(n^\omega)$ for some leading constant K_ω that is a function of the particular exponent ω and corresponding constant C_ω such that two $n \times n$ matrices can be multiplied together in

Algorithm 9: `InPlaceMM`(A, B)

Data: A an $m \times m$ matrix over a field.
Data: B an $m \times n$ matrix over a field.
Ensures: $B \leftarrow A \times B$.

```
1 begin
2     (r, [i_0, ..., i_{r-1}], P) ← CUP(A)        /* A ← [C\U] */
3     B ← PB
4     TRMM(Left, Up, Unit, A, B)                  /* B ← UB */
5     TRMM(Left, Low, NonUnit, A, B)              /* B ← CB */
6     TRLUM(A)                                    /* A ⟵ CU */
7     A ⟵ AP
```

time $C_\omega n^\omega + o(n^\omega)$. To find the leading constant K_ω of an algorithm we substituted $T(n) = K_\omega n^\omega$ into the recurrence relation for the running time. The results are summarized in Table 1, which also gives the numerical values of K_ω for the choices $(\omega, C_\omega) = (3, 2)$ and $(\omega, C_\omega) = (\log_2 7, 6)$, corresponding respectively to classical matrix multiplication and to Strassen-Winograd's algorithm (Winograd, 1971). Table 1 shows how the various rank-profile revealing elimination algorithms range in terms of time complexity: `CUP` is in $2/3n^3 + o(n^3)$, transformation to echelon form is in $n^3 + o(n^3)$, and transformation to reduced echelon form is in $2n^3 + o(n^3)$.

Figure 2 summarizes the comparison between the three approaches

- `CUP` → `ColEchTrans` → `RedColEchTrans`,
- `Gauss`, and
- `GaussJordan`

with respect to their application to solving various classical linear algebra problems.

The following observations can be made:

- Algorithm `CUP` is sufficient (i.e., the best choice in terms of time complexity) for computing the determinant, the rank, the rank profile, and the solution of a linear system: all of these invariants can be computed in time $K_\omega n^\omega + o(n^\omega)$ where K_ω is the leading constant for Algorithm `CUP`, for example $K_\omega = 2/3$ in the case where $\omega = 3$.

- Computing a transformation to echelon form (and thus a basis of the nullspace) is done by Algorithms `ColEchTrans` and `Gauss` with the same time complexity. In particular, as indicated in Table 1, the leading coefficient K_ω in the complexities of these algorithms is the same.

- Computing a transformation to reduced echelon form can be done using Algorithms `RedColEchTrans` or `GaussJordan`. Morever, since the reduced echelon form of a nonsingular matrix is the identity matrix and the corresponding transformation matrix is the inverse, these algorithms are also algorithms for matrix inversion. The two algorithms have the same leading coefficient in the time complexity for $\omega = 3$ (namely, $2n^3$), but for $\omega = \log_2 7$ Algorithm `GaussJordan` is 10% faster.

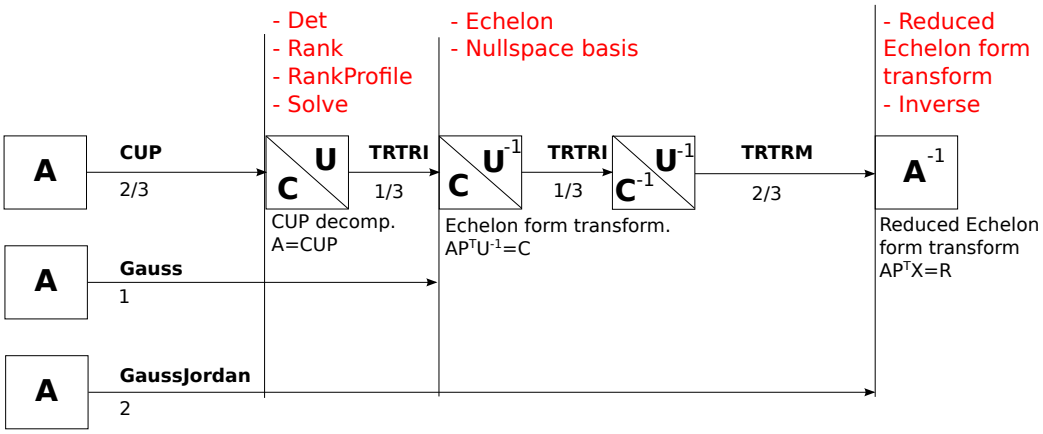


Figure 2: Application of rank revealing Gaussian elimination to linear algebra problems.

We conclude that the `CUP` decomposition is the algorithm of choice to implement rank profile revealing Gaussian elimination: it allows to compute all solutions in the best time complexity (except the case of the inverse with $\omega = \log 7$, where Algorithm `GaussJordan` is faster), either directly or using additional side computations. However, note that `GaussJordan` is not in-place; applying the technique of Algorithm 9 to make `GaussJordan` in-place increases the constant from 8 to about 21 in the case where $\omega = \log_2 7$.

Algorithm	Operation	Constant K_ω	K_3	$K_{\log_2 7}$	in-place
`MM`	$C \leftarrow AB$	C_ω	2	6	$\times$
`TRSM`	$B \leftarrow BU^{-1}$	$\frac{1}{2^{\omega-1}-2}C_\omega$	1	4	$\vee$
`TRTRI`	$U \leftarrow U^{-1}$	$\frac{1}{(2^{\omega-1}-2)(2^{\omega-1}-1)}C_\omega$	0.33	1.6	$\vee$
`TRULM`,`TRLUM`, `CUP`	$\left[L\backslash U\right] \leftarrow UL, LU; A \leftarrow \left[C\backslash U\right]$	$\left(\frac{1}{2^{\omega-1}-2} - \frac{1}{2^{\omega}-2}\right) C_\omega$	0.66	2.8	$\vee$
`ColEchTrans`, `Gauss`	$A \leftarrow \left[C\backslash U^{-1}\right]$	$\left(\frac{1}{2^{\omega-2}-1} - \frac{3}{2^{\omega}-2}\right) C_\omega$	1	4.4	$\vee$
`RedColEchTrans`	$A \leftarrow \begin{bmatrix} R\backslash T \end{bmatrix}$	$\frac{2^{\omega-1}+2}{(2^{\omega-1}-2)(2^{\omega-1}-1)}C_\omega$	2	8.8	$\vee$
`GaussJordan`	$A \leftarrow \begin{bmatrix} R\backslash T \end{bmatrix}$	$\frac{1}{2^{\omega-2}-1}C_\omega$	2	8	$\times$
`StepForm`	$A \leftarrow \left[C\backslash U^{-1}\right]$	$\left(\frac{5}{2^{\omega-1}-1} + \frac{1}{(2^{\omega-1}-1)(2^{\omega-2}-1)}\right) C_\omega$	4	15.2	$\times$

Table 1: Constants of the leading term $K_\omega n^\omega$ in the algebraic complexities for $n \times n$ invertible matrices.

5. Row echelon form and the PLE decomposition

All the algorithms and decompositions of the previous sections deal with column echelon forms that reveal the row rank profile of the input matrix. By matrix transposition, similar results can be obtained for row echelon forms and column rank profile. The natural analogue to our central tool, the CUP decomposition, is the PLE decomposition: a tuple (P, L, E) such that (E^T, L^T, P^T) is a CUP decomposition of the transposed input matrix A^T:

$$\overset{P^TA}{\begin{bmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{bmatrix}} = \overset{L}{\begin{bmatrix} 1 & & & & \\ * & 1 & & & \\ * & * & 1 & & \\ * & * & * & 1 & \\ * & * & * & & 1 \end{bmatrix}} \overset{E}{\begin{bmatrix} \overline{*_1} & * & * & * & * & * & * \\ & & \overline{*_2} & * & * & * & * \\ & & & & \overline{*_3} & * & * \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{bmatrix}}.$$

To compute a PLE decomposition one could apply Algorithm 6 with pre- and post-transpositions. A better option is to derive a "transposed" version of Algorithm 6, which directly computes a PLE decomposition in-place; as Algorithm 10 shows, this is immediate to achieve, and is what is implemented in the M4RI library (Albrecht and Bard, 2010) for dense linear algebra over GF(2). The compact rowmajor storage together with the applications driving its development (namely Gröbner basis computations (Faugère, 1999)) imposed the row echelon as a standard. The FFLAS-FFPACK (The FFLAS-FFPACK Group, 2011) library for dense linear algebra over word-size finite fields implements both the CUP and PLE Algorithms.

6. Conclusion

We have presented an algorithm computing the rank profile revealing `CUP` decomposition of a matrix over a field. We have shown that the algorithm enjoys the following features:

1. The algorithm reduces to matrix-matrix multiplications and therefore has a subcubic time complexity.
2. The complexity is rank sensitive of the form $O(mnr^{\omega-2})$, where r is the rank of the $m \times n$ input matrix.
3. The algorithm is in-place, that is, only $O(1)$ extra memory allocation for field elements is required beyond what is needed for the matrix products.
4. Used as a building block for most common computations in linear algebra, the algorithm achieves the best constant in the leading term of the time complexity. The only exception is for the reduced echelon form with $\omega = \log_2 7$, where the constant is 8.8 instead of 8 for Algorithm `GaussJordan`.

Among the set of Gaussian elimination algorithms studied here, the algorithm for `CUP` decomposition is the only one satisfying all the above conditions. For these reasons it has been chosen for the implementation of Gaussian elimination in the `FFLAS-FFPACK` and `LinBox` libraries for dense matrices over word-size finite fields (The FFLAS-FFPACK Group, 2011; The LinBox Group, 2011), as well as for the `M4RI` and `M4RIE` libraries for dense matrices over GF(2) and GF(2^e), respectively (Albrecht and Bard, 2010; Albrecht, 2011; Albrecht et al., 2011).

Appendix A. The `GaussJordan` algorithm

The `GaussJordan` algorithm (Storjohann, 2000, Algorithm 2.8) was originally presented as a fraction free algorithm for transforming a matrix over an integral domain to reduced row echelon form. We give here a simpler version of the algorithm over a field. The principle is to transform the matrix to reduced row echelon form, slice by slice. The general recursive step reduces a contiguous slice of columns of width w starting at column k by

1. recursively reducing the first half of the slice of columns (of width $w/2$),
2. updating the second half of the slice of columns (also of width $w/2$),
3. recursively reducing the second half,

Algorithm 10: PLE(A)

Data: A an $m \times n$ matrix.
Result: $(r, [i_0, \ldots, i_{r-1}], P)$.
Ensures: $A \leftarrow [L \backslash E]$ such that embedding L in an $m \times m$ unit lower triangular matrix makes $A = PLE$ a PLE decomposition of A, $r = \text{rank}(A)$ and $[i_0, \ldots, i_{r-1}]$ its column rank profile.

begin
  if $n = 1$ **then**
    if $A \neq 0$ **then**
      $i \leftarrow$ row index of the first nonzero entry of A
      $P \leftarrow T_{0,i}$, the transposition of indices 0 and i
      $A \leftarrow PA$
      for $i \leftarrow 2 \ldots m$ **do** $A_{i,0} \leftarrow A_{i,0} A_{0,0}^{-1}$
      return $(1, [0], P)$
    return $(0, [], I_n)$
  $k \leftarrow \lfloor \frac{n}{2} \rfloor$ /* $A = [A_1 | A_2]$ */
  $(r_1, [i_0, \ldots, i_{r_1-1}], P_1) \leftarrow \text{PLE}(A_{0..m}^{0..k})$ /* $A_1 \leftarrow P_1 \begin{bmatrix} L_1 \\ M_1 \end{bmatrix} E_1$ */
  if $r_1 = 0$ **then**
    $(r_2, [i_0, \ldots, i_{r_2-1}], P_2) \leftarrow \text{PLE}(A_{0..m}^{k..n})$
    return $(r_2, [i_0 + k, \ldots, i_{r_2-1} + k], P_2)$
  $A_{0..m}^{k..n} \leftarrow P_1^T A_{0..m}^{k..n}$
  /* $A = \begin{bmatrix} L_1 \backslash E_1 & A_{12} \\ M_1\ 0 & A_{22} \end{bmatrix}$ with $L_1 \backslash E_1$ $r_1 \times k$, and M_1 $(n-k) \times r_1$ */
  TRSM(Left, Low, Unit, $A_{0..r_1}^{0..r_1}, A_{0..r_1}^{k..n}$) /* $G \leftarrow L_1^{-1} A_{12}$ */
  if $r_1 = m$ **then**
    return $(r_1, [i_0, \ldots, i_{r_1-1}], P_1)$
  MM($A_{r_1..m}^{0..r_1}, A_{0..r_1}^{k..n}, A_{r_1..m}^{k..n}, -1, 1$) /* $H \leftarrow A_{22} - M_1 G$ */
  $(r_2, [j_0, \ldots, j_{r_2-1}], P_2) \leftarrow \text{PLE}(A_{r_1..m}^{k..n})$
  $A_{r_1..m}^{0..r_1} \leftarrow P_2^T A_{r_1..m}^{0..r_1}$ /* $N_1 \leftarrow P_2^T M_1$ */
  for $j \leftarrow r_1 \ldots r_1 + r_2$ **do**
    $A_{j..m}^{j} \leftarrow A_{j..m}^{j+k-r_1}$ /* Moving L_2 left next to M_1 */
    $A_{j..m}^{j+k-r_1} \leftarrow [0]$
  $P \leftarrow P_1 \text{Diag}(I_{r_1}, P_2)$
  return $(r_1 + r_2, [i_0, \ldots, i_{r_1-1}, j_0 + k, \ldots, j_{r_2-1} + k], P)$

4. composing the two transformation matrices.

The algorithm is described in full details in Algorithm 11. Calling `Gauss-Jordan`$(A, 0, 0, n)$ computes the reduced row echelon form and the associated transformation matrix. Once again, the presentation based on the indexing of the submatrices shows where all matrices are located in order to illustrate the need for extra memory allocation.

Assuming $m = n = r$, the time complexity satisfies:

$$T_{\texttt{GaussJordan}}(n, w) = 2T_{\texttt{GaussJordan}}(n, w/2) + 2T_{\texttt{MM}}(w/2, w/2, n)$$

Substituting the general form $T(n, w) = K_\omega n w^{\omega-1}$ into this recurrence relation, we obtain $K_\omega = \frac{C_\omega}{2^{\omega-1}-1}$.

Appendix B. The `StepForm` algorithm

The `StepForm` algorithm, due to Schönhage and Keller-Gehrig, is described in (Bürgisser et al., 1997, §16.5). The algorithm proceeds by first transforming the input matrix into an upper triangular matrix, and then reducing it to echelon form. Thus, the rank profile of the matrix only appears at the last step of the algorithm and the block decomposition used for the first step is not rank sensitive. As a consequence, the complexity of the algorithm does not depend on the rank of the matrix. Note that this limitation is not because of simplifying assumptions in the analysis of the complexity, but because the algorithm is itself not rank-sensitive.

We now evaluate the constant of the leading term in the complexity of the `StepForm` algorithm under the simplifying asumption $m = n = r$. The description of Algorithms Π_1, Π_2, and Π_3 used as sub-phases can be found in (Bürgisser et al., 1997, §16.5) and the references therein.

Let $T_1(n)$ be the complexity of Algorithm Π_1 applied to a $2n \times n$ matrix, and let $T_2(n)$ be the complexity of Algorithm Π_2 applied to an $n \times n$ matrix.

We have

$$T_1(n) = 4T_1(\frac{n}{2}) + 4\texttt{MM}(\frac{n}{2}, \frac{n}{2}, \frac{n}{2}) + \texttt{MM}(n, n, n),$$

from which we deduce

$$T_1(n) = C_\omega \frac{2^{\omega-2}+1}{2^{\omega-2}-1} n^\omega.$$

Algorithm 11: `GaussJordan`(A, k, s, w)

Data: A an $m \times n$ matrix over a field.
Data: k, s the column, row of the left-top coefficient of the block to reduce.
Data: w the width of the block to reduce.
Result: (r, P, Q), where P, Q are permutations matrices of order m and w.
Ensures: $r = \operatorname{rank}(A^{k..k+w}_{s..m})$ and $A \leftarrow \left[\begin{array}{c|c|c} A^{0..k}_{0..m} & \begin{matrix} X_1 & F \\ X_2 & G \\ X_3 & 0 \end{matrix} & A^{k+w..n}_{0..m} \end{array}\right]$ such that

$$\begin{bmatrix} I_s & X_1 & \\ & X_2 & \\ & X_3 & I_{m-r-s} \end{bmatrix} P A^{k..k+w}_{0..m} = \begin{bmatrix} 0 & F \\ I_r & G \\ 0 & 0 \end{bmatrix} \begin{bmatrix} I_k & \\ & Q \end{bmatrix}.$$

begin
  if $w = 1$ **then**
    if $A^k_{s..m} \neq [0]$ **then**
      $j \leftarrow$ the row index of the first nonzero entry of $A^k_{s..m}$
      $P \leftarrow T_{s,j}$ the transposition of indices s and j
      $A \leftarrow PA$
      return $(1, P, [1])$
    else
      return $(0, I_m, [1])$
  else
    $h \leftarrow \lfloor w/2 \rfloor$
    $(r_1, P_1, Q_1) =$ `GaussJordan`(A, k, s, h)
    `/* Notations:` $A = \left[\begin{array}{c|c|c|c} * & \begin{matrix} X_1 & F \\ X_2 & E \\ X_3 & \end{matrix} & \begin{matrix} Y_1 \\ Y_2 \\ Y_3 \end{matrix} & * \end{array}\right]$ `where` X_2 `is` $r_1 \times r_1$. `*/`
    $t \leftarrow s + r_1$; $g \leftarrow k + h$ `/* top left indices of` Y_3 `*/`
    MM $(A^{k..k+r_1}_{0..s}, A^{g..k+w}_{s..t}, A^{g..k+w}_{0..s}, 1, 1)$ `/*` $Y_1 \leftarrow Y_1 + X_1 Y_2$ `*/`
    $\text{temp} \leftarrow A^{g..k+w}_{s..t}$ `/* a temporary is needed for` Y_2 `*/`
    MM $(A^{k..k+r_1}_{s..t}, \text{temp}, A^{g..k+w}_{s..t}, 1, 0)$ `/*` $Y_2 \leftarrow X_2 Y_2$ `*/`
    MM $(A^{k..k+r_1}_{t..m}, A^{g..k+w}_{s..t}, A^{g..k+w}_{t..m}, 1, 1)$ `/*` $Y_3 \leftarrow Y_3 + X_3 Y_2$ `*/`
    $(r_2, P_2, Q_2) =$ `GaussJordan`$(A, g, t, w - h)$
    `/* Notations:` $A = \left[\begin{array}{c|c|c|c} * & \begin{matrix} X_1 & E_1 \\ X_2 & E_2 \\ X'_3 & \\ X'_4 & \end{matrix} & \begin{matrix} Z_1 & F_1 \\ Z_2 & F_2 \\ Z_3 & F_3 \\ Z_4 & \end{matrix} & * \end{array}\right]$ `with` Z_3 $r_2 \times r_2$ `*/`
    MM $(A^{g..g+r_2}_{0..t}, A^{k..k+r_1}_{t..t+r_2}, A^{k..k+r_1}_{0..t}, 1, 1)$ `/*` $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \leftarrow \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} X'_3$ `*/`
    $\text{temp} \leftarrow A^{k..k+r_1}_{t..t+r_2}$ `/* a temporary is needed for` X'_3 `*/`
    MM $(A^{g..g+r_2}_{t..t+r_2}, temp, A^{k..k+r_1}_{t..t+r_2}, 1, 0)$ `/*` $X'_3 \leftarrow Z_3 X'_3$ `*/`
    MM $(A^{g..g+r_2}_{t+r_2..m}, A^{k..k+r_1}_{t..t+r_2}, A^{k..k+r_1}_{t+r_2..m}, 1, 1)$ `/*` $X'_4 \leftarrow X'_4 + Z_4 X'_3$ `*/`
    $Q \leftarrow \begin{bmatrix} I_{r_1} & & & \\ & I_{h-r_1} & & \\ & & I_{r_2} & \\ & & & I_{w-h-r_2} \end{bmatrix} \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix}$
    return $(r_1 + r_2, P_2 P_1, Q)$

We have

$$\begin{aligned} T_2(n) &= 2T_2(\frac{n}{2}) + T_1(\frac{n}{2}) + 5\mathtt{MM}(\frac{n}{2}, \frac{n}{2}, \frac{n}{2}) + \mathtt{MM}(n, n, \frac{n}{2}) \\ &= 2T_2(\frac{n}{2}) + C_\omega \left(\frac{2^{\omega-2}+1}{2^{\omega-2}-1} + \frac{9}{2^\omega} \right) n^\omega \end{aligned}$$

from which we deduce

$$T_2(n) = C_\omega \left(\frac{5}{2^{\omega-1}-1} + \frac{1}{(2^{\omega-1}-1)(2^{\omega-2}-1)} \right) n^\omega.$$

Under our assumption $n = m = r$, Algorithm Π_3 does not perform any operation and the total complexity is therefore $T_2(n)$. For $(\omega, C_\omega) = (3, 2)$ we obtain $T_2(n) = 4n^3$, and for $(\omega, C_\omega) = (\log_2 7, 6)$ we obtain $T_2(n) = \frac{76}{5} n^{\log_2 7} = 15.2n^{2.81}$.

Finally, the algorithm does not specify that the transformation matrix generated by algorithm Π_2 is lower triangular: it needs to be stored in some additional memory space and thus the algorithm is not in-place.

References

Aho, A., Hopcroft, J., Ullman, J., 1974. The Design and Analysis of Computer Algorithms. Addison-Wesley.

Albrecht, M., Bard, G., 2010. The M4RI Library – Version 20100817. The M4RI Team.

Albrecht, M.R., 2011. The M4RIE library for dense linear algebra over small fields with even characteristic. available at http://arxiv.org/abs/1111.6900. Submitted for publication.

Albrecht, M.R., Bard, G.V., Pernet, C., 2011. Efficient dense Gaussian elimination over the finite field with two elements. available at http://arxiv.org/abs/1111.6549. Submitted for publication.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D., 1999. LAPACK Users' Guide. SIAM, Philadelphia, PA. third edition.

Boyer, B., Dumas, J.G., Pernet, C., Zhou, W., 2009. Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm, in: Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC), Seoul, Korea, ACM. pp. 55–62.

Bunch, J., Hopcroft, J., 1974. Triangular factorization and inversion by fast matrix multiplication. Mathematics of Computation 28, 231–236.

Bürgisser, B., Clausen, C., Shokrollahi, M., 1997. Algebraic Complexity Theory. volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag.

Corless, R.M., Jeffrey, D.J., 1997. The Turing factorization of a rectangular matrix. SIGSAM Bull. 31, 20–30.

Dongarra, J.J., Croz, J.D., Hammarling, S., Duff, I.S., 1990. A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. 16, 1–17.

Dumas, J.G., Gautier, T., Pernet, C., 2002. Finite field linear algebra subroutines, in: Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC), Lille, France, ACM. pp. 63–74.

Dumas, J.G., Giorgi, P., Pernet, C., 2008. Dense linear algebra over word-size prime fields: the FFLAS and FFPACK packages. ACM Trans. Math. Softw. 35, 1–42.

Faugère, J.C., 1999. A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra 139, 61–88.

von zur Gathen, J., Gerhard, J., 2003. Modern Computer Algebra. Cambridge University Press. second edition.

Golub, G., Van Loan, C., 1996. Matrix Computations. The Johns Hopkins University Press. third edition.

Huss-Lederman, S., Jacobson, E.M., Johnson, J.R., Tsao, A., Turnbull, T., 1996. Strassen's Algorithm for Matrix Multiplication : Modeling Analysis, and Implementation. Technical Report. Center for Computing Sciences. CCS-TR-96-17.

Ibarra, O., Moran, S., Hui, R., 1982. A generalization of the fast LUP matrix decomposition algorithm and applications. Journal of Algorithms 3, 45–56.

Jeannerod, C.P., 2006. LSP matrix decomposition revisited. Technical Report RR2006-28. INRIA - LIP, ENS Lyon.

Jeffrey, D.J., 2010. LU factoring of non-invertible matrices. SIGSAM Bull. 44, 1–8.

Keller-Gehrig, W., 1985. Fast algorithms for the characteristic polynomial. Theoretical Computer Science 36, 309–317.

Pilgrim, M., 2004. Dive into Python.

Schönhage, A., 1973. Unitäre Transformationen großer Matrizen. Numerische Mathematik 20, 409–417.

Stein, W., 2007. Modular forms, a computational approach. Graduate studies in mathematics, American Mathematical Society.

Storjohann, A., 2000. Algorithms for Matrix Canonical Forms. Ph.D. thesis. Swiss Federal Institute of Technology, ETH-Zurich.

Storjohann, A., Mulders, T., 1998. Fast algorithms for linear algebra modulo N, in: Gianfranco Bilardi, Giuseppe F. Italiano, A.P., Pucci, G. (Eds.), Algorithms — ESA'98, Springer. pp. 139–150.

Strassen, V., 1969. Gaussian elimination is not optimal. Numerische Mathematik 13, 354–356.

The FFLAS-FFPACK Group, 2011. The FFLAS-FFPACK Library (version 1.4.2).

The LinBox Group, 2011. The LinBox Library (Version 1.2.1). `http://linalg.org`.

Turing, A., 1948. Rounding-off errors in matrix processes. Quart. J. Mech. Appl. Math. 1, 287–308.

Winograd, S., 1971. On multiplication of 2×2 matrices. Linear Algebra Appl. 4, 381–388.